Machine Learning Engineer Nanodegree

Capstone Project

Christopher Martinez

November 18 2018

I. Definition

Project Overview

This project is based on the Google Analytics Customer Revenue Prediction competition detailed here: https://www.kaggle.com/c/ga-customer-revenue-prediction

It is worth mentioning that it is based on the first version of the data (v1) and not the current (v2).

The 80/20 rule has proven true for many businesses–only a small percentage of customers produce most of the revenue. As such, marketing teams are challenged to make appropriate investments in promotional strategies. [1]

The goal of the competition is to demonstrate how important is machine learning and data analysis for companies, by predicting the total amount spent on the Google Merchandise Store for each customer. It will be very helpful for every company to better identify and target the type of customer that is expending more money on their business given that most of the people visiting their business are not going to purchase anything, so it will be wasteful to spend resource targeting them.

Problem Statement

A very few percentage of all the people who visit a website do not buy anything, or goal here is to predict how much a visitor will spend on the google store based on some data set. By doing so we will improve our target marketing and focus it on those customers who are likely to spend more money on the site.

We need to predict the the natural log of the sum of all transactions per user using a pre given training set of ninety thousand rows by fourteen columns.

$$y_{user} = \sum_{i=1}^{n} transaction_{user_i}$$

$$target_{user} = \ln(y_{user} + 1)$$

We need to predict the the natural log of the sum of all transactions per user using a pre given training set of ninety thousand rows by fourteen columns.

We will get familiar with the data and plot some graphs in order to identify which features could be important and which are of no use to us. That information will help us to create if needed new features and transform those, like in this case apply a log transformation to our target. We will need to do the same for categorical features.

We will try different machine learning algorithms for regression, since we are trying to predict a continuous target on the dataset measuring their effectiveness on the mean squared error, mean absolute error and training time:

- ADAboostregressor
- GradientBoostingregressor
- XGBregressor

Boosters are vastly used by the kaggle community for this kinds of supervised learning problems, they are robust, powerful and proved, that is why all the algorithms to be tested are boosters.

Once we find the best performer we will tune it using Bayesian Optimization, which is a computationally cheap technique of hyperparameter tuning compared to the Grid Search but much more accurate compared to the Random Search, both which I was not very eager to use. The untuned model will be the benchmark to beat based solely on the RMSE score. Finally we will make some predictions on the testing set.

Metrics

Since this is a **regression** problem we will use 2 metrics for the algorithm testing but only one for the improvement, tuning and final testing. We are using the MAE just as a tie-breaker in case our algorithms have very close RMSE scores and similar training time.

Root Mean Squared Error (RMSE): We want to be as accurate as possible with all our predictions, so we want to heavily punish large errors in our testing set even if they do not occur as much, the squaring of the difference between the prediction and the target help us to and punish those errors.

I am using the RMSE mainly because in the metric the competition's poster asked us to use and it makes sense to me, because what they want is a robust model applicable to all visits to the store so we need to be truly aware of outliers and punish them greatly, therefore RMSE will do the trick.

This will be the only metric to measure our final model compared to our benchmark.

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2},$$

More specifically we will use scikit learn metrics module to measure the RMSE.

Mean Absolute Error(MAE): It measures the overall error across the predictions, we will only use it when testing models in case we get close results in the other metrics, it will not be use to beat the benchmark, only to pick the best algorithm.

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

The MAE just will be a tie-breaker in case of close scores in the testing but will not be of any use in other parts of the process, since our main goal is to pick those outliers and predict every visitor well.

We will also use scikit learn metrics module to measure the MAE.

II. Analysis

Data Exploration

You can download all the datasets used in this project in the following links:

https://www.dropbox.com/s/o8aqviril0xqrvn/data.zip?dl=0

https://www.dropbox.com/s/a40y2yiknobs6ct/Data-flattened.zip?dl=0

We have a dataset of approximately 90 thousand rows , each corresponding to a single visit to the store[2]. Each row has the next columns:

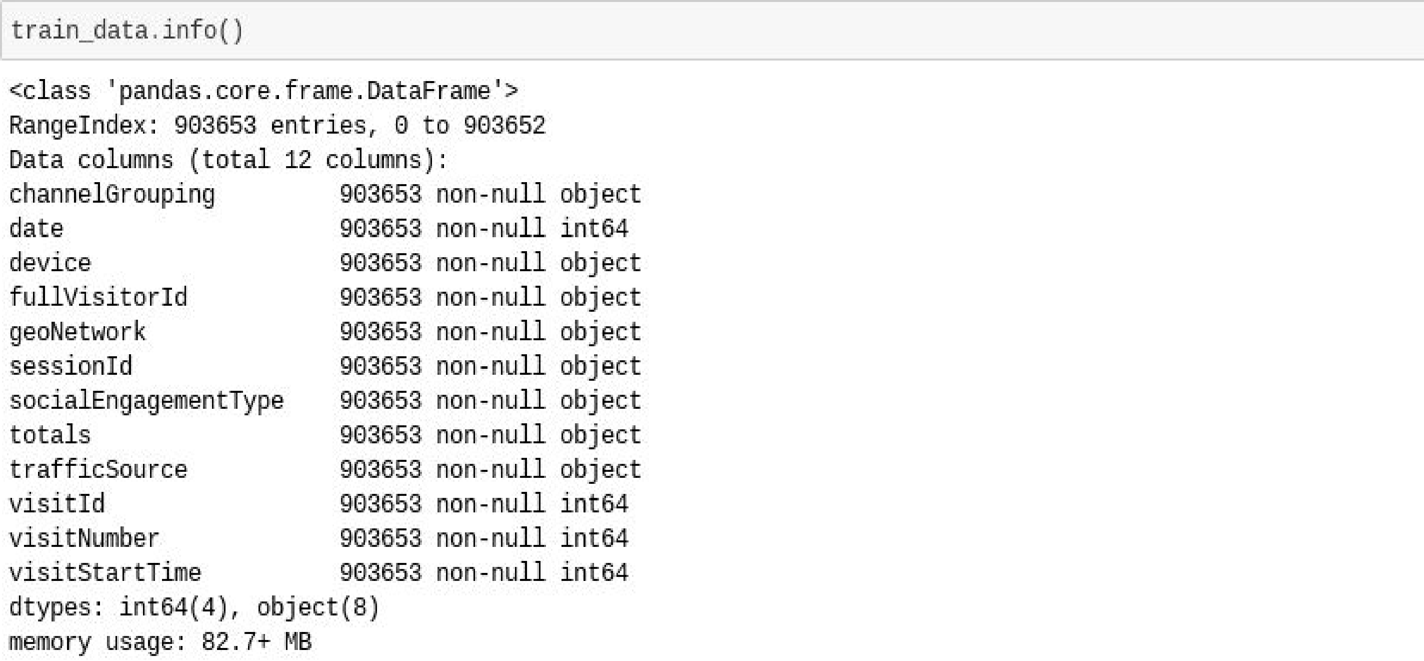
1. fullVisitorId- A unique identifier for each user of the Google Merchandise Store.
2. channelGrouping - The channel via which the user came to the Store.
3. date - The date on which the user visited the Store.
4. device - The specifications for the device used to access the Store.
5. geoNetwork - This section contains information about the geography of the user.
6. socialEngagementType - Engagement type, either "Socially Engaged" or "Not Socially Engaged".
7. totals - This section contains aggregate values across the session.
8. trafficSource - This section contains information about the Traffic Source from which the session originated.
9. visitId - An identifier for this session. This is part of the value usually stored as the _utmb cookie. This is only unique to the user. For a completely unique ID, you should use a combination of fullVisitorId and visitId.
10. visitNumber - The session number for this user. If this is the first session, then this is set to 1.

11. visitStartTime - The timestamp (expressed as POSIX time).

12. hits - This row and nested fields are populated for any and all types of hits. Provides a record of all page visits.

13. customDimensions - This section contains any user-level or session-level custom dimensions that are set for a session. This is a repeated field and has an entry for each dimension that is set.

14. totals - This set of columns mostly includes high-level aggregate data.

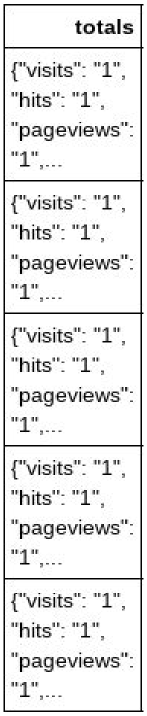


```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 903653 entries, 0 to 903652
Data columns (total 12 columns):
channelGrouping         903653 non-null object
date                    903653 non-null int64
device                  903653 non-null object
fullVisitorId           903653 non-null object
geoNetwork              903653 non-null object
sessionId               903653 non-null object
socialEngagementType    903653 non-null object
totals                  903653 non-null object
trafficSource           903653 non-null object
visitId                 903653 non-null int64
visitNumber             903653 non-null int64
visitStartTime          903653 non-null int64
dtypes: int64(4), object(8)
memory usage: 82.7+ MB
```

There are multiple columns which contain JSON blobs of varying depth. In one of those JSON columns, totals, the sub-column ‘transactionRevenue’ contains the revenue information we are trying to predict. We used this notebook to do the parsing:

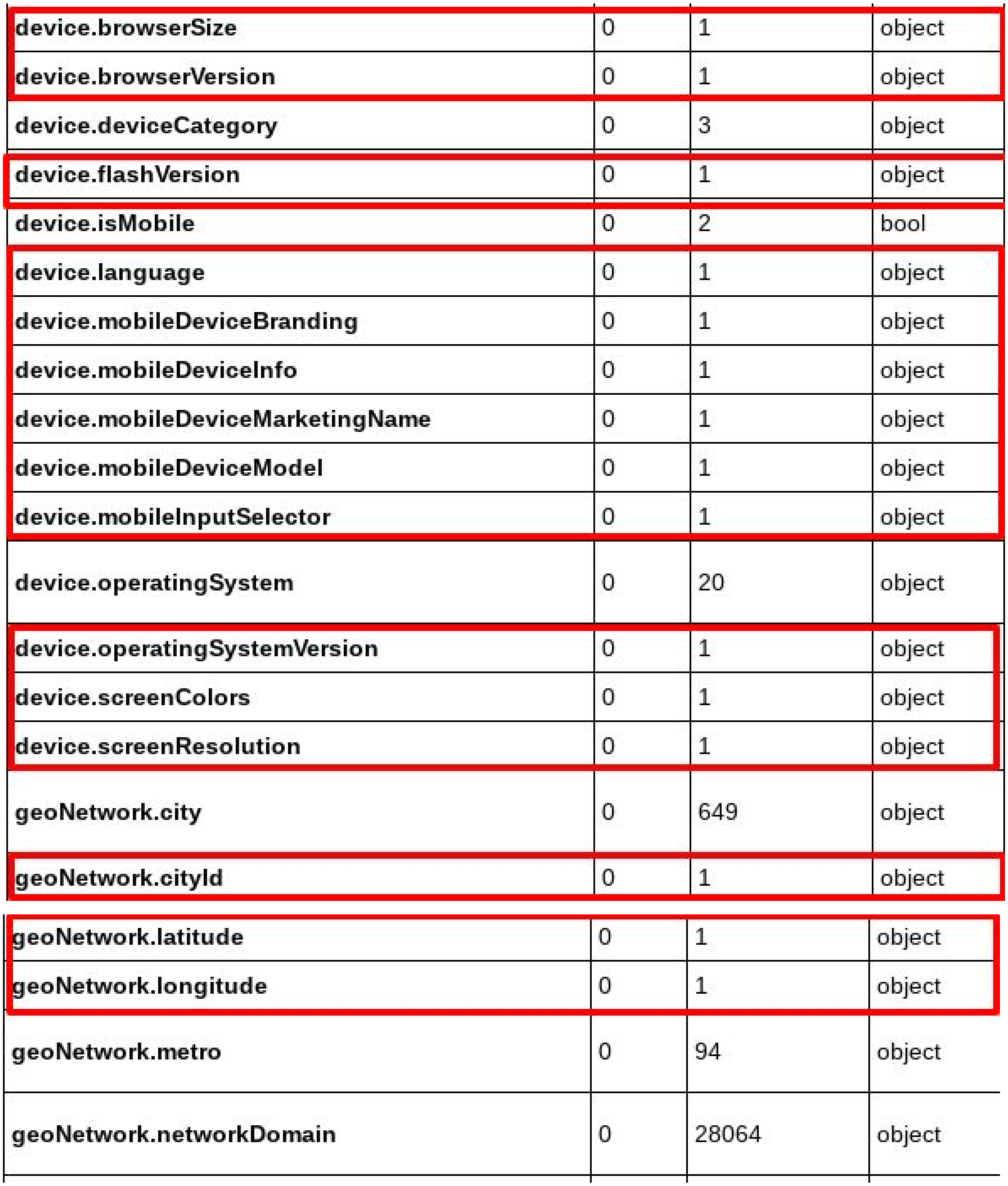
https://www.kaggle.com/julian3833/1-quick-start-read-csv-and-flatten-json-fields



totals
{"visits": "1", "hits": "1", "pageviews": "1",...
{"visits": "1", "hits": "1", "pageviews": "1",...
{"visits": "1", "hits": "1", "pageviews": "1",...
{"visits": "1", "hits": "1", "pageviews": "1",...
{"visits": "1", "hits": "1", "pageviews": "1",...

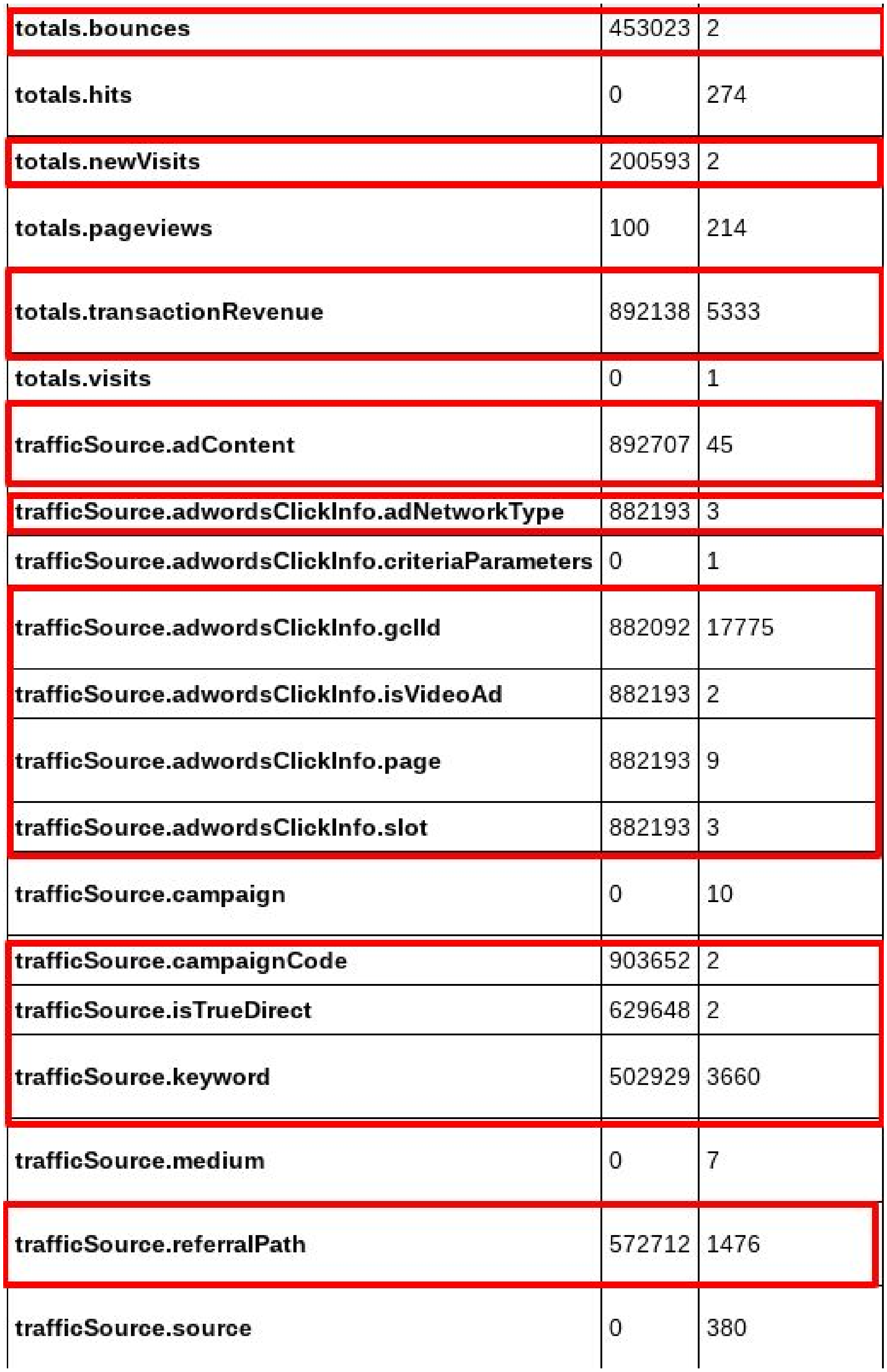
After the parsing we have identified some columns with the constant value of ‘ NaN’ , that date is not available to us and therefore those columns are useless so we need to get rid of the next columns:

	Null	Unique_Count	Data_type



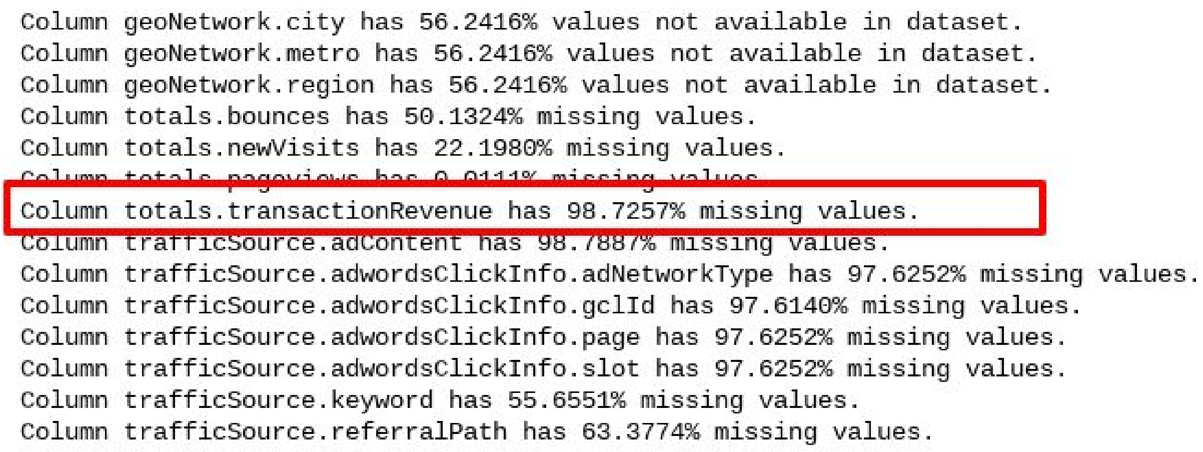
device.browserSize	0	1	object
device.browserVersion	0	1	object
device.deviceCategory	0	3	object
device.flashVersion	0	1	object
device.isMobile	0	2	bool
device.language	0	1	object
device.mobileDeviceBranding	0	1	object
device.mobileDeviceInfo	0	1	object
device.mobileDeviceMarketingName	0	1	object
device.mobileDeviceModel	0	1	object
device.mobileInputSelector	0	1	object
device.operatingSystem	0	20	object
device.operatingSystemVersion	0	1	object
device.screenColors	0	1	object
device.screenResolution	0	1	object
geoNetwork.city	0	649	object
geoNetwork.cityId	0	1	object
geoNetwork.latitude	0	1	object
geoNetwork.longitude	0	1	object
geoNetwork.metro	0	94	object
geoNetwork.networkDomain	0	28064	object

After doing a summary analysis of our dataset we realized that we are dealing with a great deal of missing values, especially in the next columns:

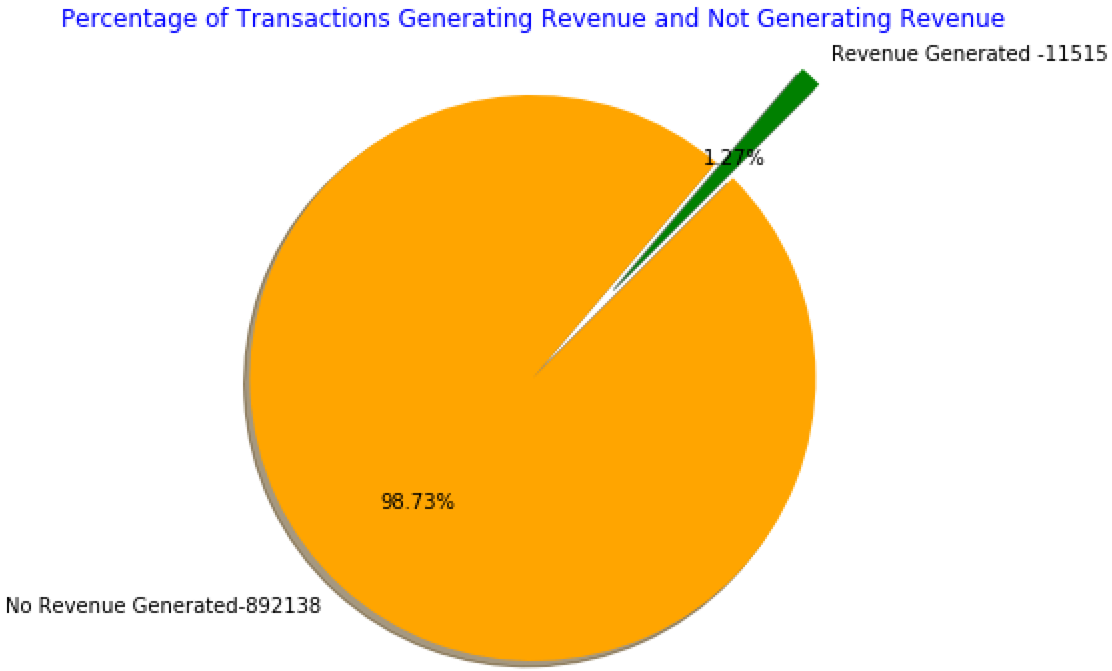


totals.bounces	453023	2
totals.hits	0	274
totals.newVisits	200593	2
totals.pageviews	100	214
totals.transactionRevenue	892138	5333
totals.visits	0	1
trafficSource.adContent	892707	45
trafficSource.adwordsClickInfo.adNetworkType	882193	3
trafficSource.adwordsClickInfo.criteriaParameters	0	1
trafficSource.adwordsClickInfo.gclId	882092	17775
trafficSource.adwordsClickInfo.isVideoAd	882193	2
trafficSource.adwordsClickInfo.page	882193	9
trafficSource.adwordsClickInfo.slot	882193	3
trafficSource.campaign	0	10
trafficSource.campaignCode	903652	2
trafficSource.isTrueDirect	629648	2
trafficSource.keyword	502929	3660
trafficSource.medium	0	7
trafficSource.referralPath	572712	1476
trafficSource.source	0	380

The first statement in the competition just came to be true as far as our analysis goes, only about 2 % of the transaction had generated any revenue.



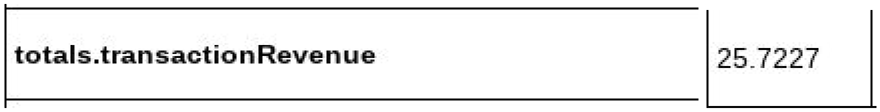
```
Column geoNetwork.city has 56.2416% values not available in dataset.
Column geoNetwork.metro has 56.2416% values not available in dataset.
Column geoNetwork.region has 56.2416% values not available in dataset.
Column totals.bounces has 50.1324% missing values.
Column totals.newVisits has 22.1980% missing values.
Column totals.pageviews has 0.0111% missing values.
Column totals.transactionRevenue has 98.7257% missing values.
Column trafficSource.adContent has 98.7887% missing values.
Column trafficSource.adwordsClickInfo.adNetworkType has 97.6252% missing values.
Column trafficSource.adwordsClickInfo.gclId has 97.6140% missing values.
Column trafficSource.adwordsClickInfo.page has 97.6252% missing values.
Column trafficSource.adwordsClickInfo.slot has 97.6252% missing values.
Column trafficSource.keyword has 55.6551% missing values.
Column trafficSource.referralPath has 63.3774% missing values.
```



The parasing also left us with a lot of categorical features to deal with:

‘device.browser', 'device.deviceCategory', 'device.isMobile', 'device.operatingSystem', 'geoNetwork.city', 'geoNetwork.continent', 'geoNetwork.country', 'geoNetwork.metro', 'geoNetwork.networkDomain','geoNetwork.region','geoNetwork.subContinent',trafficSource.campaign', 'trafficSource.isTrueDirect', 'trafficSource.keyword','trafficSource.medium', 'trafficSource.referralPath','trafficSource.source'.

Checking for the skewness in our features we realized why we need to get the natural log of the total revenue. The other seem to have an acceptable level of skewness:

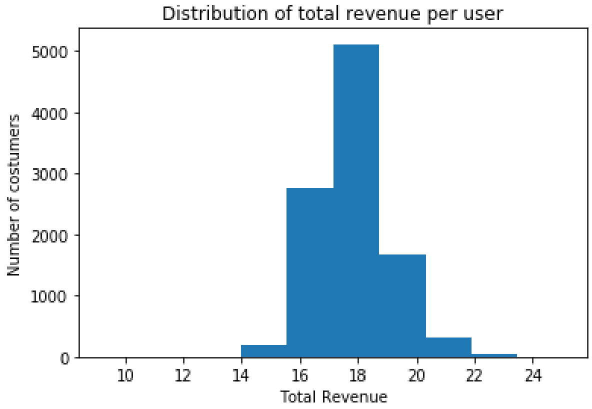


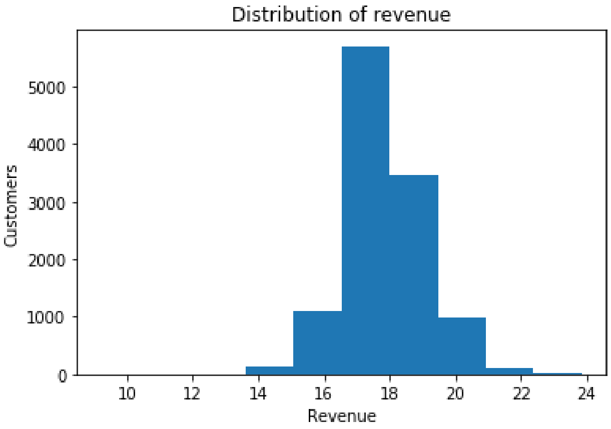
totals.transactionRevenue	25.7227

Exploratory Visualization

Revenue distributions

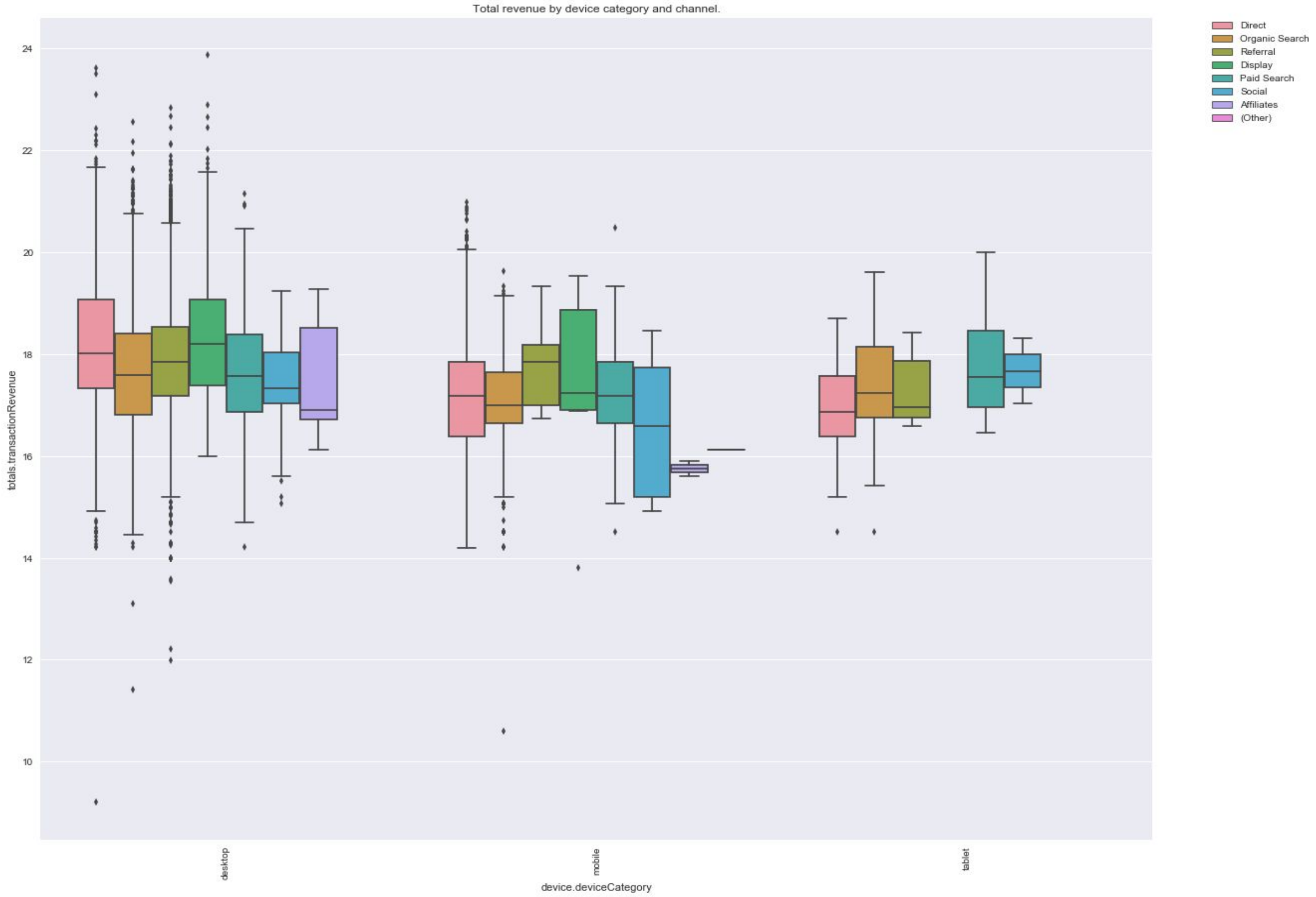
There is almost no difference between the distribution of revenue and the total distribution of revenue per user.





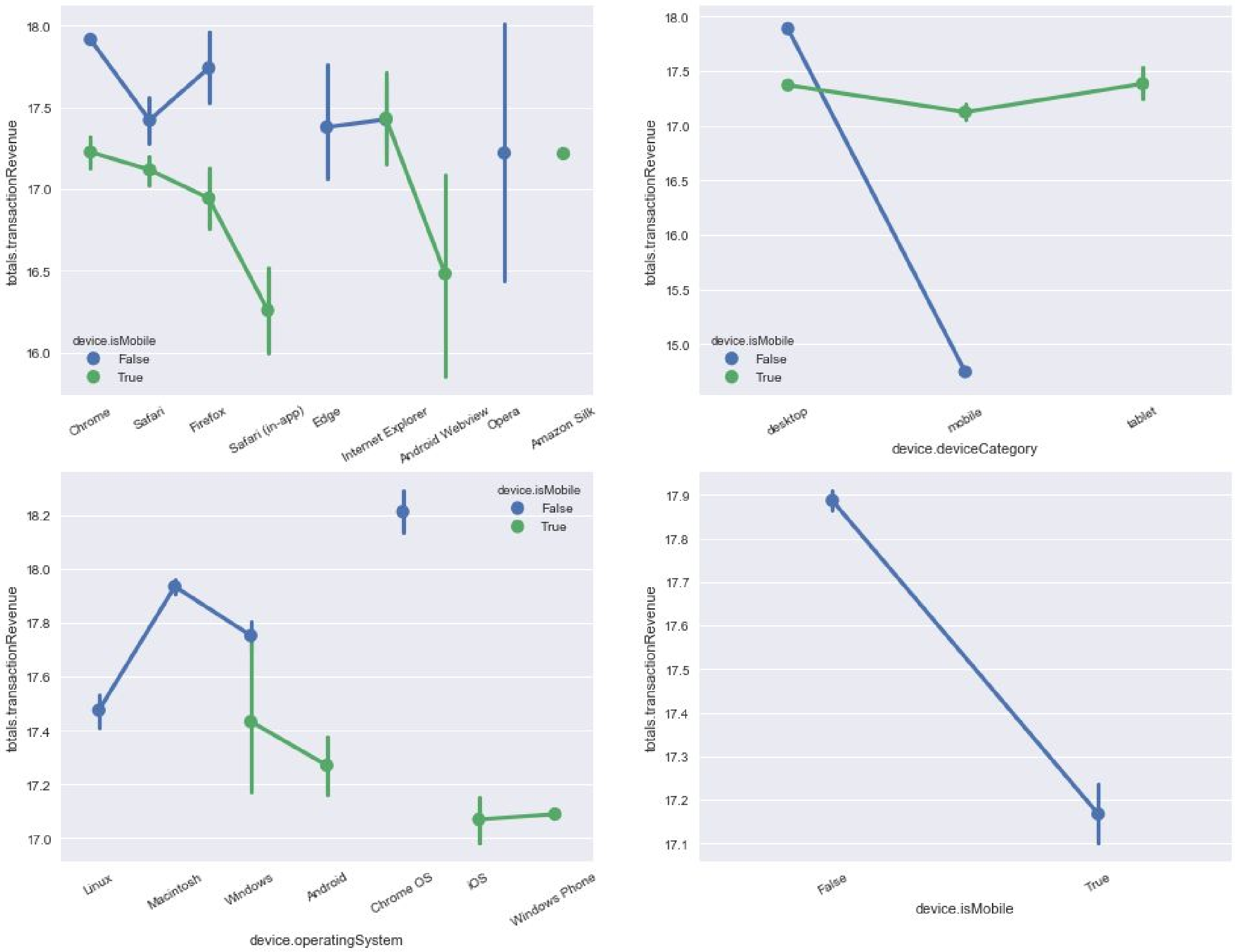
Where does the paying users come from?

We can see that the desktop is the prefered devices for our users to make purchases and that the least profitable channels are Affiliate and Others.



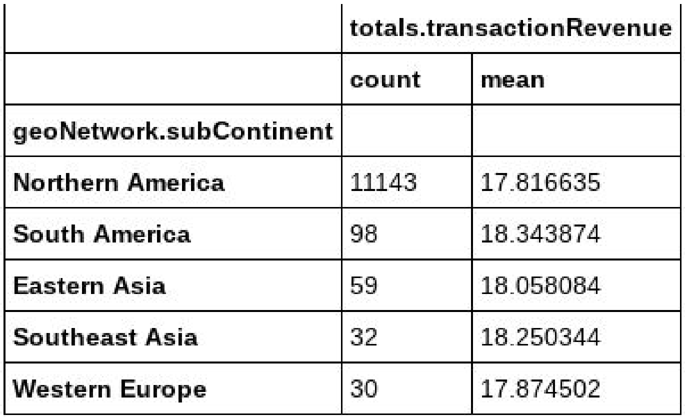
Devices

Chrome OS and Macintosh are clear winners here.



geoNetwork

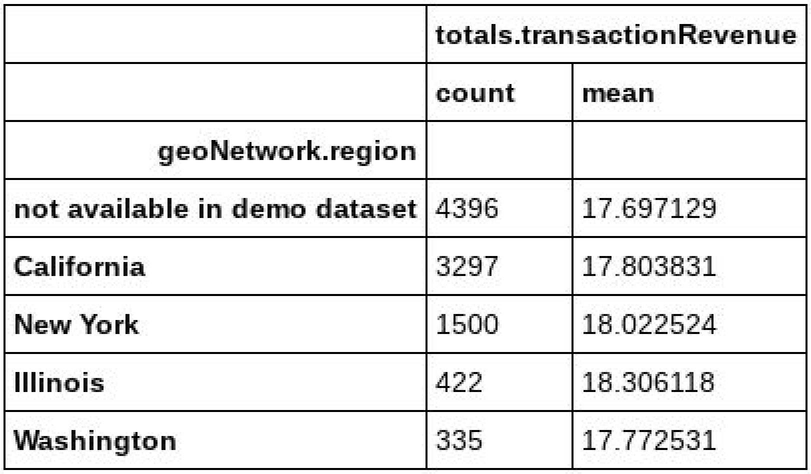
Is not a surprise that most of the revenue comes from North America.



	totals.transactionRevenue	
	count	mean
geoNetwork.subContinent		
Northern America	11143	17.816635
South America	98	18.343874
Eastern Asia	59	18.058084
Southeast Asia	32	18.250344
Western Europe	30	17.874502

Which regions?¶

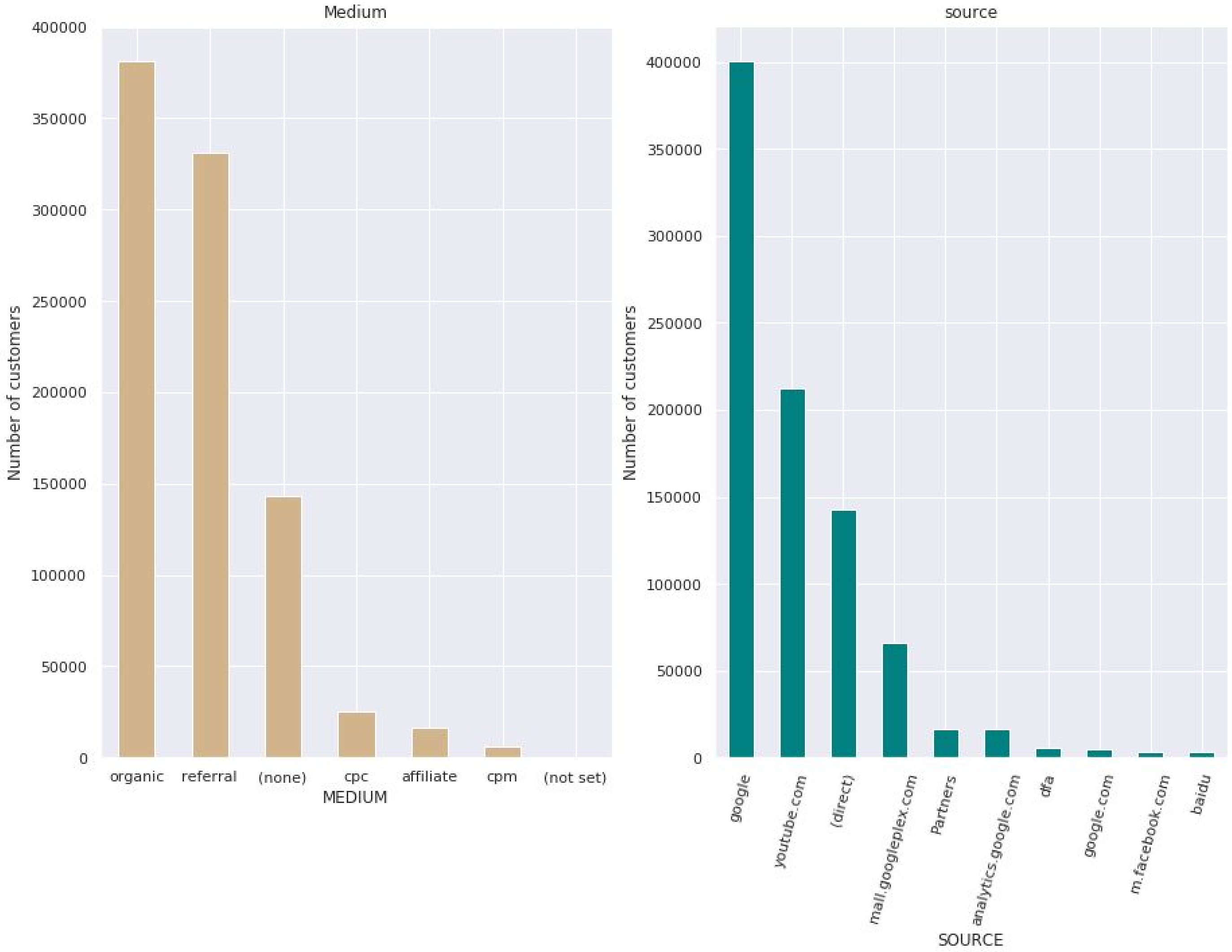
California and New York get the golden medals here.



	totals.transactionRevenue	
	count	mean
geoNetwork.region		
not available in demo dataset	4396	17.697129
California	3297	17.803831
New York	1500	18.022524
Illinois	422	18.306118
Washington	335	17.772531

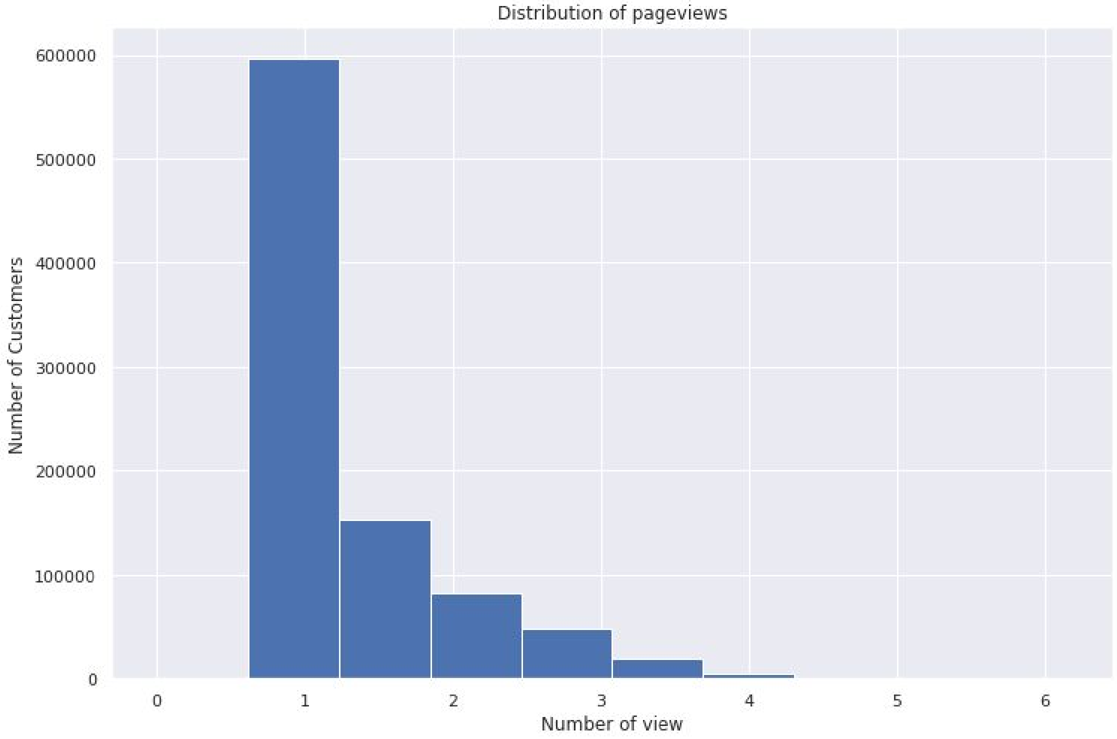
Traffic source

Google related sources are really on the head here, as well as organic and referral.

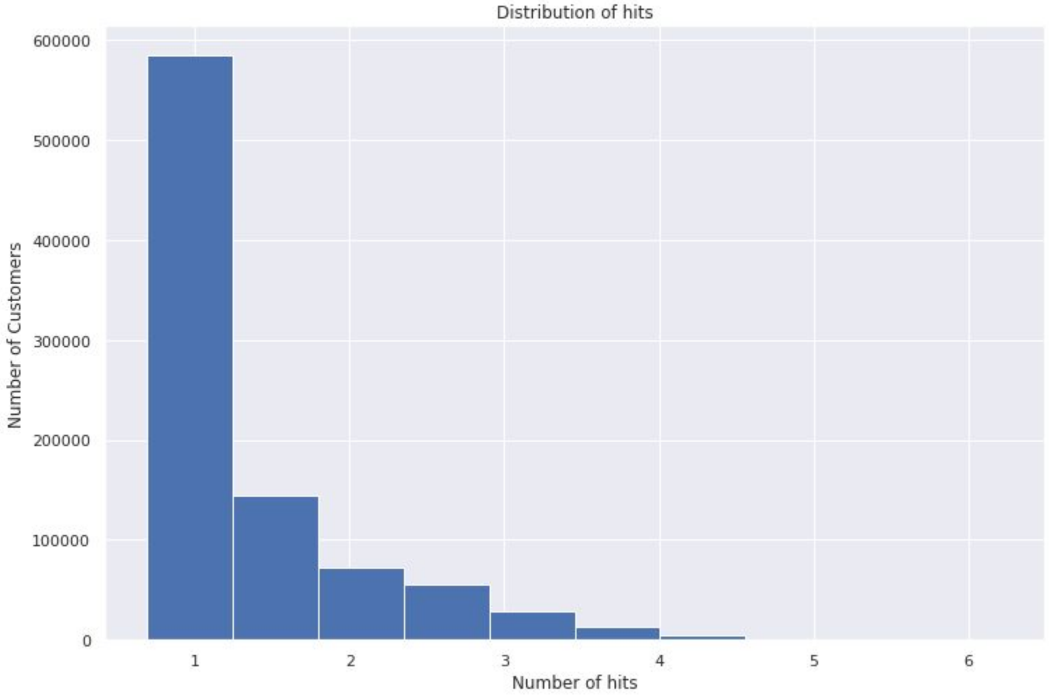

Medium
Number of customers
400000
350000
300000
250000
200000
150000
100000
50000
0
organic
referral
(none)
cpc
affiliate
cpm
(not set)
MEDIUM
source
Number of customers
400000
350000
300000
250000
200000
150000
100000
50000
0
google
youtube.com
(direct)
mall.googleplex.com
Partners
analytics.google.com
dfa
google.com
m.facebook.com
baidu
SOURCE

Totals

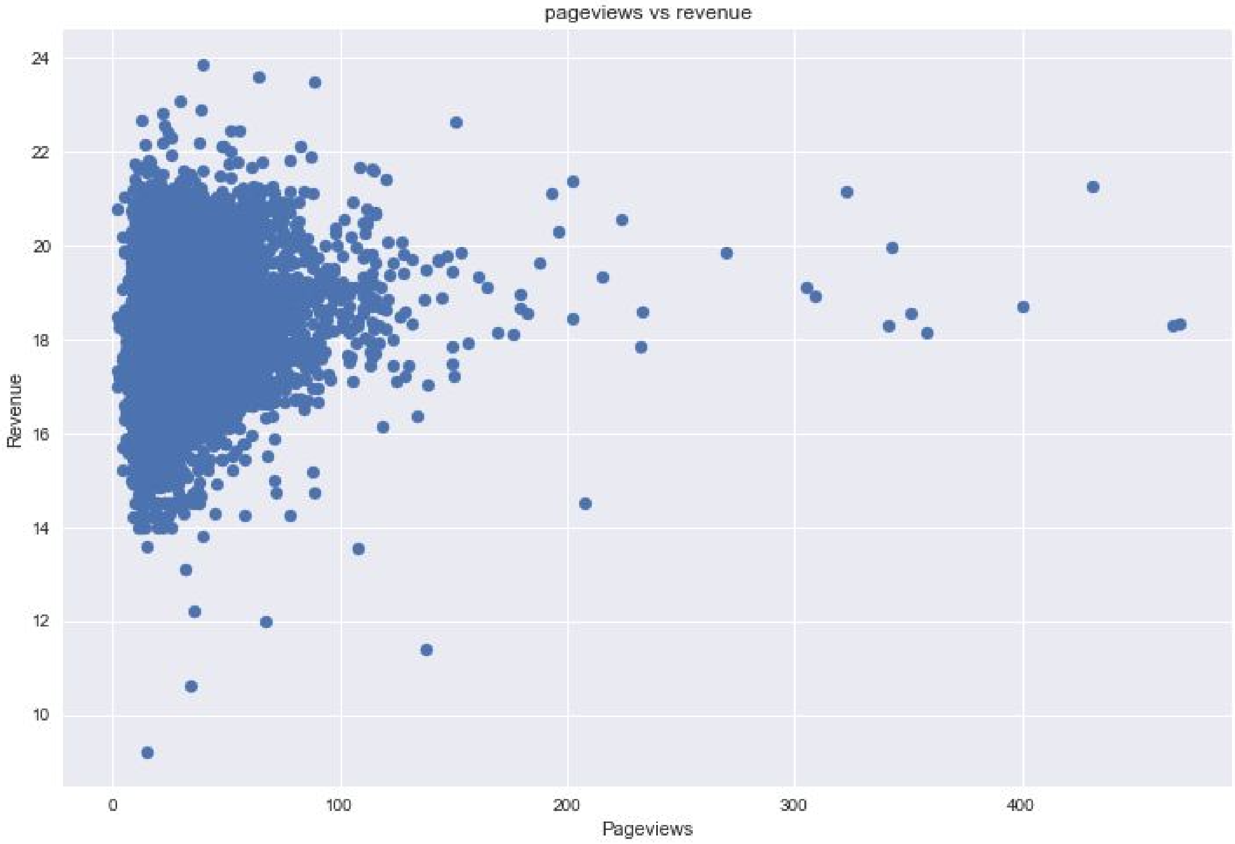
Most of the visit only view the page once.



The same goes for number of hits, most visitors are not staying in page a lot.



How are pageview relates to revenue?



There is some correlation with the number of times a customers visits the page and the amount of revenue they create, the more visits the more they spend.

Algorithms and Techniques

The machine learning algorithms we will try in our dataset are as follow:

- XGBregressor:

It is an implementation of gradient boosting machines created by Tianqi Chen, now with contributions from many developers. It belongs to a broader collection of tools under the umbrella of the Distributed Machine Learning Community or DMLC who are also the creators of the popular mxnet deep learning library. [3]

A lot of people just say that the main purpose of XGBoost is to win Kaggle competitions, so it was an obvious choice to at least test it, here are the pros and cons:

Pros:

- It is used a lot by kagglers. It is fast. It is versatile.

Cons:

- If not tuned correctly it is likely to overfit. Only takes numerical features so a lot of preprocessing is required.

Personal opinion:

- I one of the most famous algorithms out there, I haven't tried anytime before so I really would love to use it.

https://xgboost.readthedocs.io/en/latest/parameter.html

- ADAboosregressor:

AdaBoost, short for Adaptive Boosting, is a machine learning meta-algorithm formulated by Yoav Freund and Robert Schapire, who won the 2003 Gödel Prize for their work. It can be used in conjunction with many other types of learning algorithms to improve performance. The output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier. [4]

Pros:

- Less susceptible to the overfitting problem than most learning algorithms. Easy to use and tweak.

Cons:

- It can be very susceptible to noisy data.

Personal opinion:

- I have tried the classification version of this algorithm and it worked marvelous, I think it is a good candidate to at least be tested. Old but gold.

We will use a regressor version of the algorithm available in scikit learn:

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.html

- GradientBoostingRegressor:

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.[5]

- Pros:

Flexible yet powerful algorithm.

- Cons:

Parameter sensitive, computationally expensive and can overfit in not tune correctly.

- Personal opinion:

I was really curious how it compared to his brother the XGBboost. It was my first choice for the problem before I read about XGBboost.

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

- Metrics.

We will use the RMSE , the MAE and how much time each one takes to train our dataframe.

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2},$$

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_absolute_error.html

We will use the best performer on those regards as our benchmark model and try to improve it using a hypermapmert tuning technique called Bayesian Optimization.

- Bayesian Optimization overview:

Bayesian Optimization is a constrained global optimization package built upon bayesian inference and gaussian process, that attempts to find the maximum value of an unknown function in as few iterations as possible. This technique is particularly suited for optimization of high cost functions, situations where the balance between exploration and exploitation is important.[6] http://philipperemy.github.io/visualization/

- Pros:

Faster than grid search, computationally cheap, easy to use and proved by several kagglers.

- Cons:

Relatively new, has a hard time with big dimensions and if not used correctly it can be worse than random search.

- Personal opinion:

I did not have the time for grid search and wanted to try something new within the project.

- Bayesian Optimization module:

Bayesian optimization works by constructing a posterior distribution of functions (gaussian process) that best describes the function you want to optimize. As the number of observations grows, the posterior distribution improves, and the algorithm becomes more certain of which regions in parameter space are worth exploring and which are not, as seen in the picture below.[7]

https://github.com/fmfn/BayesianOptimization

Bayesian Optimization application notebook:

I follow this tutorial on how to implement bayesian optimization in a XGBregressor algorithm:

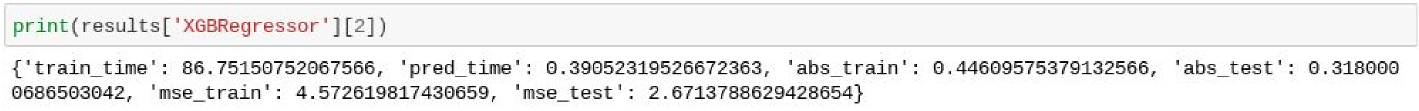
https://www.kaggle.com/btyuhas/bayesian-optimization-with-xgboost

Benchmark

As explained above our benchmark model will be the untuned algorithm with better performance in RMSE, MAE and training time. But our goal will be to outperformed this model in RMSE using parameter tuning by Bayesian Optimization.

This is our benchmark score:

RMSE Test: 2.6713
RMSE Train: 4.5726



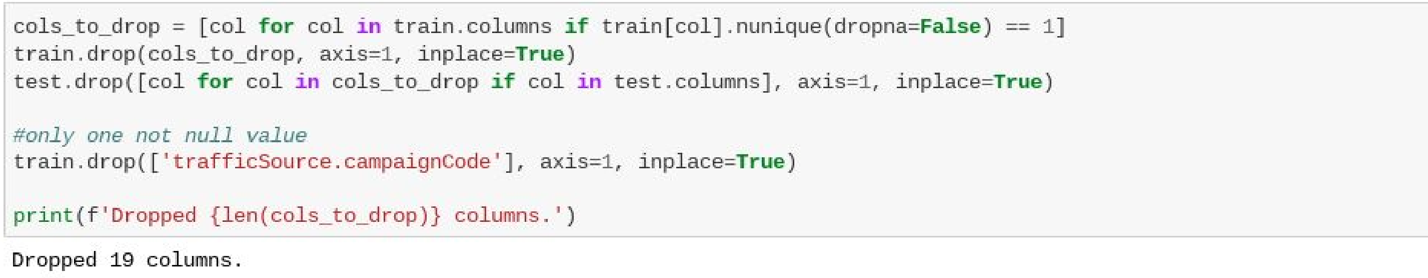
```
print(results['XGBRegressor'][2])
```

```
{'train_time': 86.75150752067566, 'pred_time': 0.39052319526672363, 'abs_train': 0.44609575379132566, 'abs_test': 0.318000
0686503042, 'mse_train': 4.572619817430659, 'mse_test': 2.6713788629428654}
```

III. Methodology

Data Preprocessing

We already have gotten rid of this useless columns:



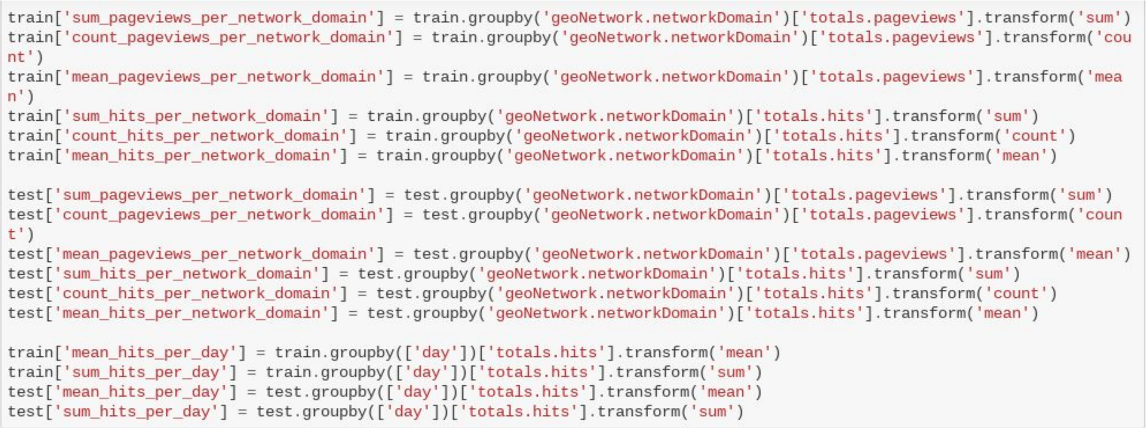
```
cols_to_drop = [col for col in train.columns if train[col].nunique(dropna=False) == 1]
train.drop(cols_to_drop, axis=1, inplace=True)
test.drop([col for col in cols_to_drop if col in test.columns], axis=1, inplace=True)

#only one not null value
train.drop(['trafficSource.campaignCode'], axis=1, inplace=True)

print(f'Dropped {len(cols_to_drop)} columns.')
```

```
Dropped 19 columns.
```

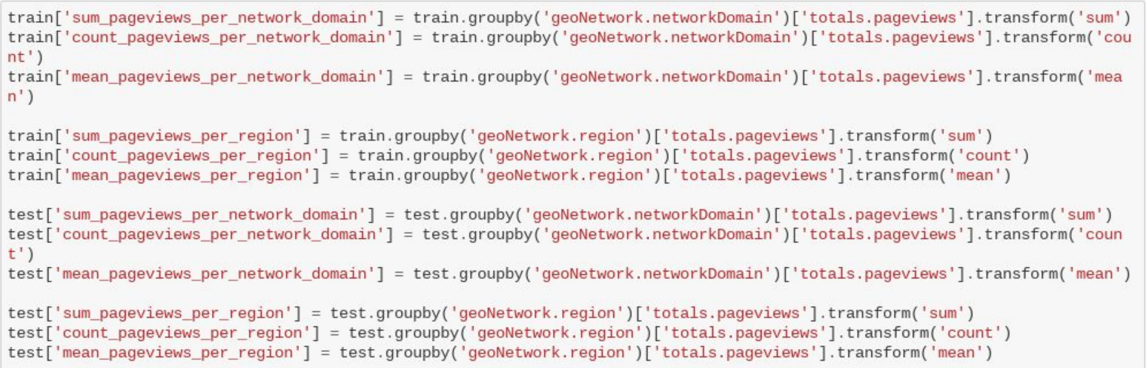
We need to do a little bit of feature engineering. We will use the date to get us the week, day, month and week of year for every transaction and we are going to get the mean, sum and unique count of the Totals, Device and GeoNetwork flatten JSON fields, as we believe this features will have an important impact in our model:



```
train['sum_pageviews_per_network_domain'] = train.groupby('geoNetwork.networkDomain')['totals.pageviews'].transform('sum')
train['count_pageviews_per_network_domain'] = train.groupby('geoNetwork.networkDomain')['totals.pageviews'].transform('cou
nt')
train['mean_pageviews_per_network_domain'] = train.groupby('geoNetwork.networkDomain')['totals.pageviews'].transform('mea
n')
train['sum_hits_per_network_domain'] = train.groupby('geoNetwork.networkDomain')['totals.hits'].transform('sum')
train['count_hits_per_network_domain'] = train.groupby('geoNetwork.networkDomain')['totals.hits'].transform('count')
train['mean_hits_per_network_domain'] = train.groupby('geoNetwork.networkDomain')['totals.hits'].transform('mean')

test['sum_pageviews_per_network_domain'] = test.groupby('geoNetwork.networkDomain')['totals.pageviews'].transform('sum')
test['count_pageviews_per_network_domain'] = test.groupby('geoNetwork.networkDomain')['totals.pageviews'].transform('coun
t')
test['mean_pageviews_per_network_domain'] = test.groupby('geoNetwork.networkDomain')['totals.pageviews'].transform('mean')
test['sum_hits_per_network_domain'] = test.groupby('geoNetwork.networkDomain')['totals.hits'].transform('sum')
test['count_hits_per_network_domain'] = test.groupby('geoNetwork.networkDomain')['totals.hits'].transform('count')
test['mean_hits_per_network_domain'] = test.groupby('geoNetwork.networkDomain')['totals.hits'].transform('mean')

train['mean_hits_per_day'] = train.groupby(['day'])['totals.hits'].transform('mean')
train['sum_hits_per_day'] = train.groupby(['day'])['totals.hits'].transform('sum')
test['mean_hits_per_day'] = test.groupby(['day'])['totals.hits'].transform('mean')
test['sum_hits_per_day'] = test.groupby(['day'])['totals.hits'].transform('sum')
```



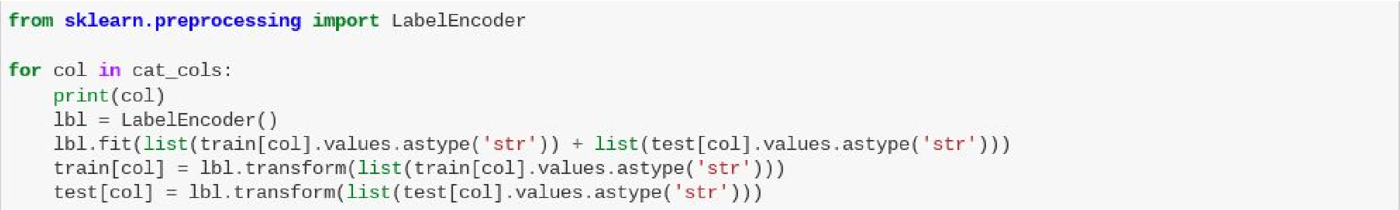
```
train['sum_pageviews_per_network_domain'] = train.groupby('geoNetwork.networkDomain')['totals.pageviews'].transform('sum')
train['count_pageviews_per_network_domain'] = train.groupby('geoNetwork.networkDomain')['totals.pageviews'].transform('cou
nt')
train['mean_pageviews_per_network_domain'] = train.groupby('geoNetwork.networkDomain')['totals.pageviews'].transform('mea
n')

train['sum_pageviews_per_region'] = train.groupby('geoNetwork.region')['totals.pageviews'].transform('sum')
train['count_pageviews_per_region'] = train.groupby('geoNetwork.region')['totals.pageviews'].transform('count')
train['mean_pageviews_per_region'] = train.groupby('geoNetwork.region')['totals.pageviews'].transform('mean')

test['sum_pageviews_per_network_domain'] = test.groupby('geoNetwork.networkDomain')['totals.pageviews'].transform('sum')
test['count_pageviews_per_network_domain'] = test.groupby('geoNetwork.networkDomain')['totals.pageviews'].transform('coun
t')
test['mean_pageviews_per_network_domain'] = test.groupby('geoNetwork.networkDomain')['totals.pageviews'].transform('mean')

test['sum_pageviews_per_region'] = test.groupby('geoNetwork.region')['totals.pageviews'].transform('sum')
test['count_pageviews_per_region'] = test.groupby('geoNetwork.region')['totals.pageviews'].transform('count')
test['mean_pageviews_per_region'] = test.groupby('geoNetwork.region')['totals.pageviews'].transform('mean')
```

We are going to deal with the categorical features using a label encoder to transform them into numbers:



```
from sklearn.preprocessing import LabelEncoder

for col in cat_cols:
    print(col)
    lbl = LabelEncoder()
    lbl.fit(list(train[col].values.astype('str')) + list(test[col].values.astype('str')))
    train[col] = lbl.transform(list(train[col].values.astype('str')))
    test[col] = lbl.transform(list(test[col].values.astype('str')))
```

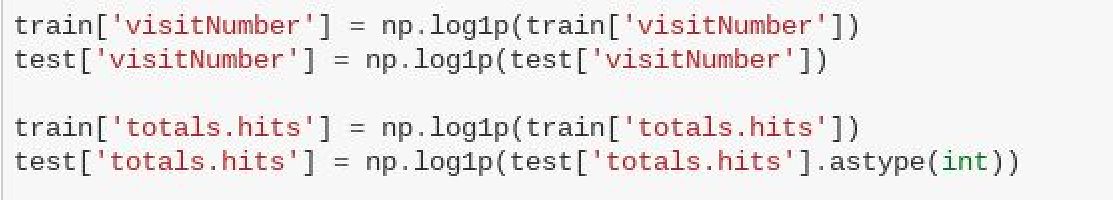
Labelencoder:

And to convert this kind of categorical text data into model-understandable numerical data, we use the Label Encoder class. So all we have to do, to label encode the first column, is import the LabelEncoder class from the sklearn library, fit and transform the first column of the data, and then replace the existing text data with the new encoded data.

We used this module to get the LabelEncoder class:

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

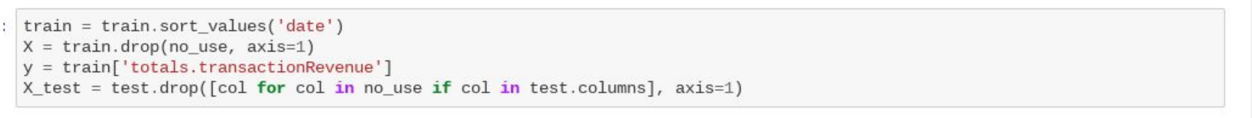
Logarithmic transformation in numerical features to avoid skewness :



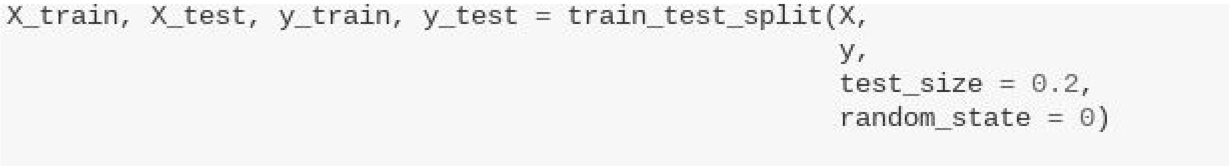
```
train['visitNumber'] = np.log1p(train['visitNumber'])
test['visitNumber'] = np.log1p(test['visitNumber'])

train['totals.hits'] = np.log1p(train['totals.hits'])
test['totals.hits'] = np.log1p(test['totals.hits'].astype(int))
```

Now we will get our target y and our features X and divide the data into training and testing sets to test our algorithms and pick the best performer.



```
train = train.sort_values('date')
X = train.drop(no_use, axis=1)
y = train['totals.transactionRevenue']
X_test = test.drop([col for col in no_use if col in test.columns], axis=1)
```



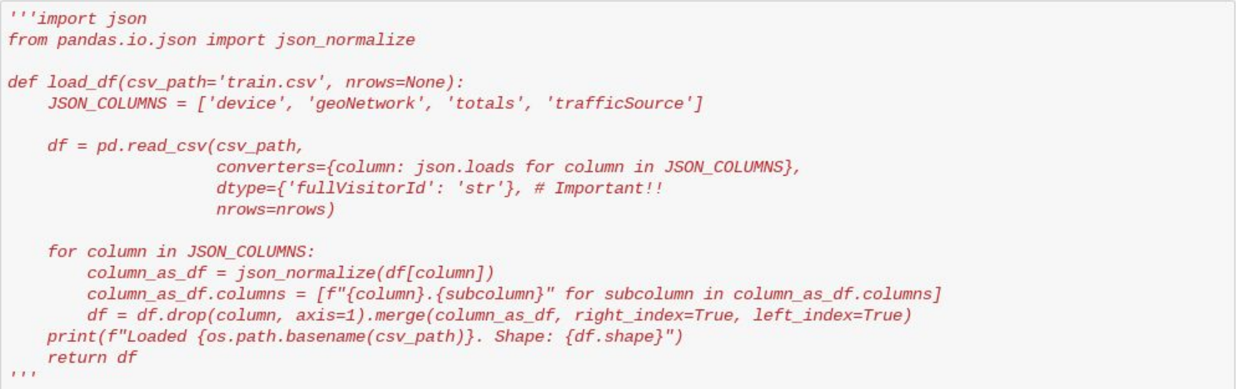
```
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size = 0.2,
                                                    random_state = 0)
```

Implementation

We took the next steps and implemented the algorithms and techniques along the way in the next order:

1. Data exploration

Firstly we needed to parse some JSON fields contained in our data, we used the next function to do so, it was kindly provided by Julian :



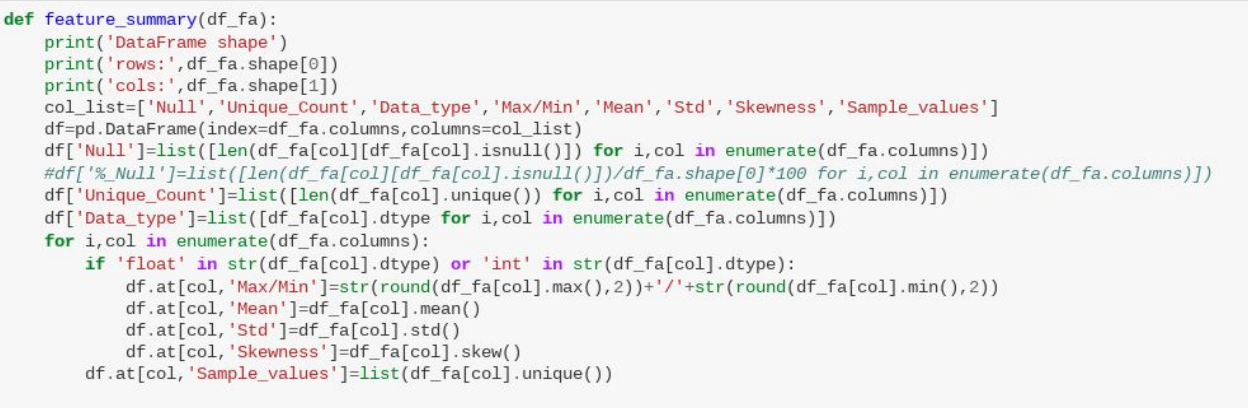
```
'''import json
from pandas.io.json import json_normalize

def load_df(csv_path='train.csv', nrows=None):
    JSON_COLUMNS = ['device', 'geoNetwork', 'totals', 'trafficSource']

    df = pd.read_csv(csv_path,
                     converters={column: json.loads for column in JSON_COLUMNS},
                     dtype={'fullVisitorId': 'str'}, # Important!!
                     nrows=nrows)

    for column in JSON_COLUMNS:
        column_as_df = json_normalize(df[column])
        column_as_df.columns = [f"{column}.{subcolumn}" for subcolumn in column_as_df.columns]
        df = df.drop(column, axis=1).merge(column_as_df, right_index=True, left_index=True)
    print(f"Loaded {os.path.basename(csv_path)}. Shape: {df.shape}")
    return df
'''
```

For the exploration we used several pandas and numpy functions to explore the data, specially we used a function created inspired by the Kaggle community called:



```
def feature_summary(df_fa):
    print('DataFrame shape')
    print('rows:',df_fa.shape[0])
    print('cols:',df_fa.shape[1])
    col_list=['Null','Unique_Count','Data_type','Max/Min','Mean','Std','Skewness','Sample_values']
    df=pd.DataFrame(index=df_fa.columns,columns=col_list)
    df['Null']=list([len(df_fa[col][df_fa[col].isnull()]) for i,col in enumerate(df_fa.columns)])
    #df['%_Null']=list([len(df_fa[col][df_fa[col].isnull()])/df_fa.shape[0]*100 for i,col in enumerate(df_fa.columns)])
    df['Unique_Count']=list([len(df_fa[col].unique()) for i,col in enumerate(df_fa.columns)])
    df['Data_type']=list([df_fa[col].dtype for i,col in enumerate(df_fa.columns)])
    for i,col in enumerate(df_fa.columns):
        if 'float' in str(df_fa[col].dtype) or 'int' in str(df_fa[col].dtype):
            df.at[col,'Max/Min']=str(round(df_fa[col].max(),2))+'/'+str(round(df_fa[col].min(),2))
            df.at[col,'Mean']=df_fa[col].mean()
            df.at[col,'Std']=df_fa[col].std()
            df.at[col,'Skewness']=df_fa[col].skew()
        df.at[col,'Sample_values']=list(df_fa[col].unique())
```

Our goal was to get familiar with the data, check for useless features and for features that will need certain transformations in the next step.

2. Data preprocessing:

   We created some features of our own using pre existing ones.

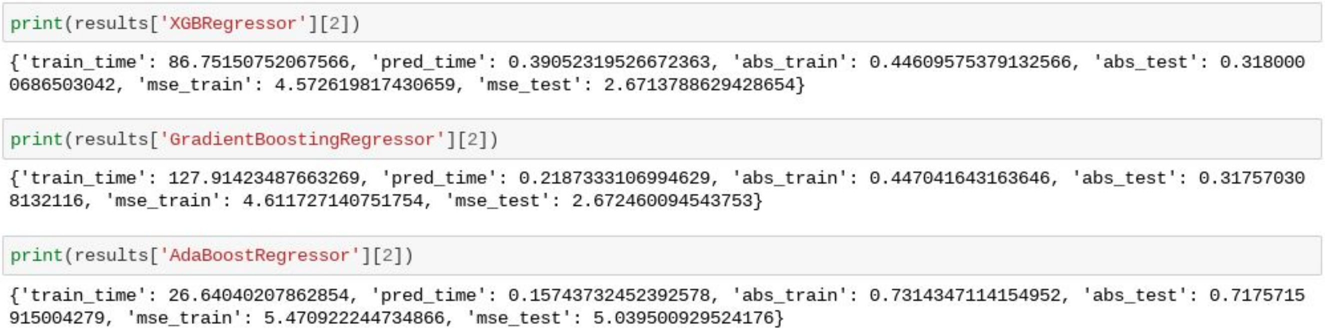
   We use a labelencoder to transform our categorical features into numbers .

   We got the natural log of some feature in order to reduce skewness and because the main problem asked to predict the natural log of total.Revenue.

We defined our target and our final features and uses scikit learn to split them into training and testing sets and targets.

3. Model evaluation:

   We took some inspiration from the code given to us in the ‘ Donors prediction project’ to evaluate our algorithms, we changed the metrics of course and used RMSE and AME to evaluate the three algorithm mentioned above:



```
print(results['XGBRegressor'][2])
```

```
{'train_time': 86.75150752067566, 'pred_time': 0.39052319526672363, 'abs_train': 0.44609575379132566, 'abs_test': 0.318000
0686503042, 'mse_train': 4.572619817430659, 'mse_test': 2.6713788629428654}
```

```
print(results['GradientBoostingRegressor'][2])
```

```
{'train_time': 127.91423487663269, 'pred_time': 0.2187333106994629, 'abs_train': 0.447041643163646, 'abs_test': 0.31757030
8132116, 'mse_train': 4.611727140751754, 'mse_test': 2.672460094543753}
```

```
print(results['AdaBoostRegressor'][2])
```

```
{'train_time': 26.64040207862854, 'pred_time': 0.15743732452392578, 'abs_train': 0.7314347114154952, 'abs_test': 0.7175715
915004279, 'mse_train': 5.470922244734866, 'mse_test': 5.039500929524176}
```

   It is important to mention that we used the default parameters for all three models, there were no changes in that regard, they are just the standard.

   XGBregressor and GadientBoostin were pretty close in RMSE and AMS but since a lot of the Kaggle community is being using the first one and because it was a bit better on training time he have decided to use it as pour benchmark and later try to improve it using hypermarapeter tuning by Bayesian Optimization.

4. Model improvement:

   Instead of using the usual Grid search approach we went for the more computationally, time friendly and in my personal opinion better approach of Bayesian Optimization, we took a lot of inspiration from this notebook on how to use it on XGBregressor and we used the Bayesian Optimization module that can be found here:

   Module

   Notebook

   Difficulties along the way:

   The greatest difficulty was the model parsing, I was really lucky to come across a kaggler who already did it for me, anyways there was a problem, that some of the parsed columns have a ‘.’in their names, so it was difficult to condanet certain functions because of the point, and had to use [ ] when calling certain methods for those columns.

Refinement

I tuned the XGBregressor parameters using Bayesian Optimization.

Bayesian optimization is hyperparameter tuning technique that samples some input-outputs (less than 10) and use them to guess the true function with something called a 'Gaussian Process' . Then use that guessed function to determine where to evaluate next. Evaluate that point, add it to our set of input-outputs and infer the guessed function once again. Repeat this until you've exhausted your budget of evaluations. We will try to maximize our ‘eval-metric’, since that would make little to no sense we need to target the negative RMSE.

Tuning the Bayesian Optimization algorithm is an art more than a science so I followed this notebook tutorial and then play a little bit myself until getting the best results. I used the following parameters:

- num_boost_round=2000
- nfold=5
- init_points=10
- n_iter=10
- acq='ei

A class in grid search is too computationally expensive for me to do and random sampling is an ide that I have never liked, just hoping to get lucky, so Bayesian optimization was my way to go, these are the rage of parameters we tuned:

```
'max_depth': (3,5),
'gamma': (2,5),
'min_child_weight': (.5,2.0),
'max_delta_step': (0.01,.5),
'subsample': (0.0,1.0),
'colsample_bytree' :(.1,1),
 'eta':(.001,.1)})
```

These are the best results:

{'max_depth': 5, 'gamma': 2.0, 'min_child_weight': 2.0, 'max_delta_step': 0.5, 'subsample': 0.9, 'colsample_bytree': 1.0, 'eta': 0.1}

It took about 2:15 hrs to get to those parameters.

IV. Results

Model Evaluation and Validation

After a few hours of training we got the next RMSE scores in testing and in training:

Testing: 1.6483372591347045

Training: 1.5045276725190662

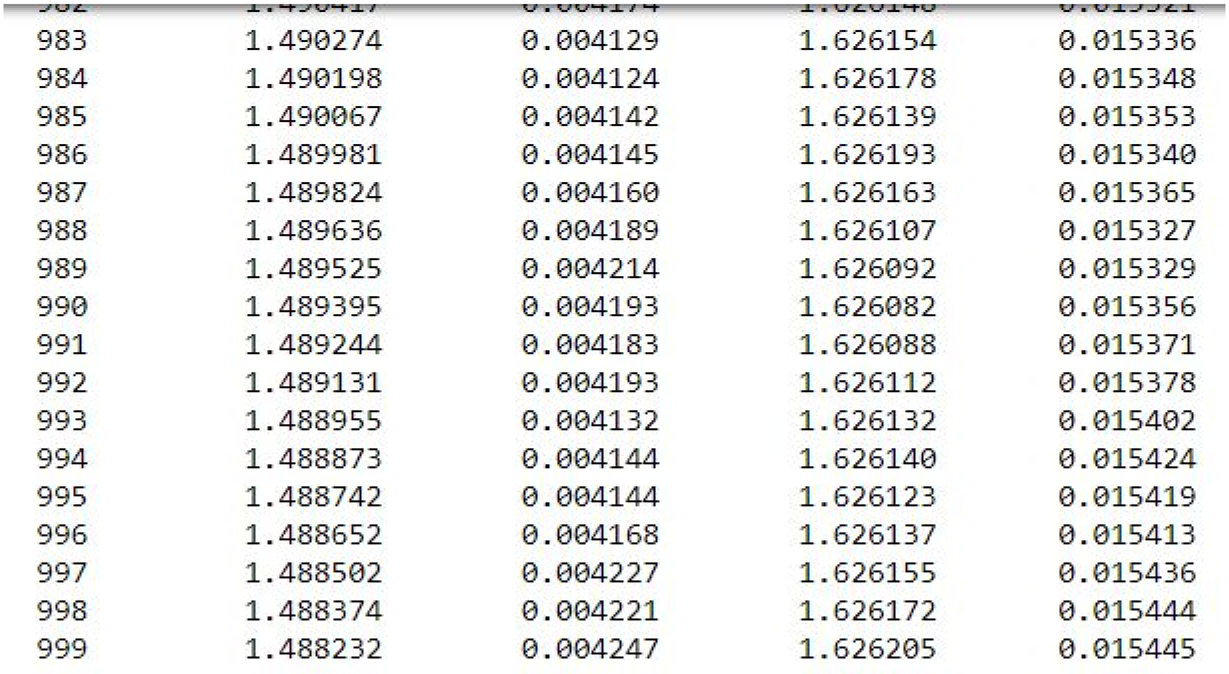
Compared to other Kagglers who get an average of 1.65 to 1.7 in their first try suing grid search and with similar features, our model did amazingly good and took significantly less time to be improved.

Justification

If we compare it to our benchmark model, we can see the huge difference in performance:

Metrics	Benchmark	Tuned model
Training RMSE	2.6713788629428654	1.5045276725190662
Testing RMSE	4.572619817430659	1.6483372591347045

Cross validation.



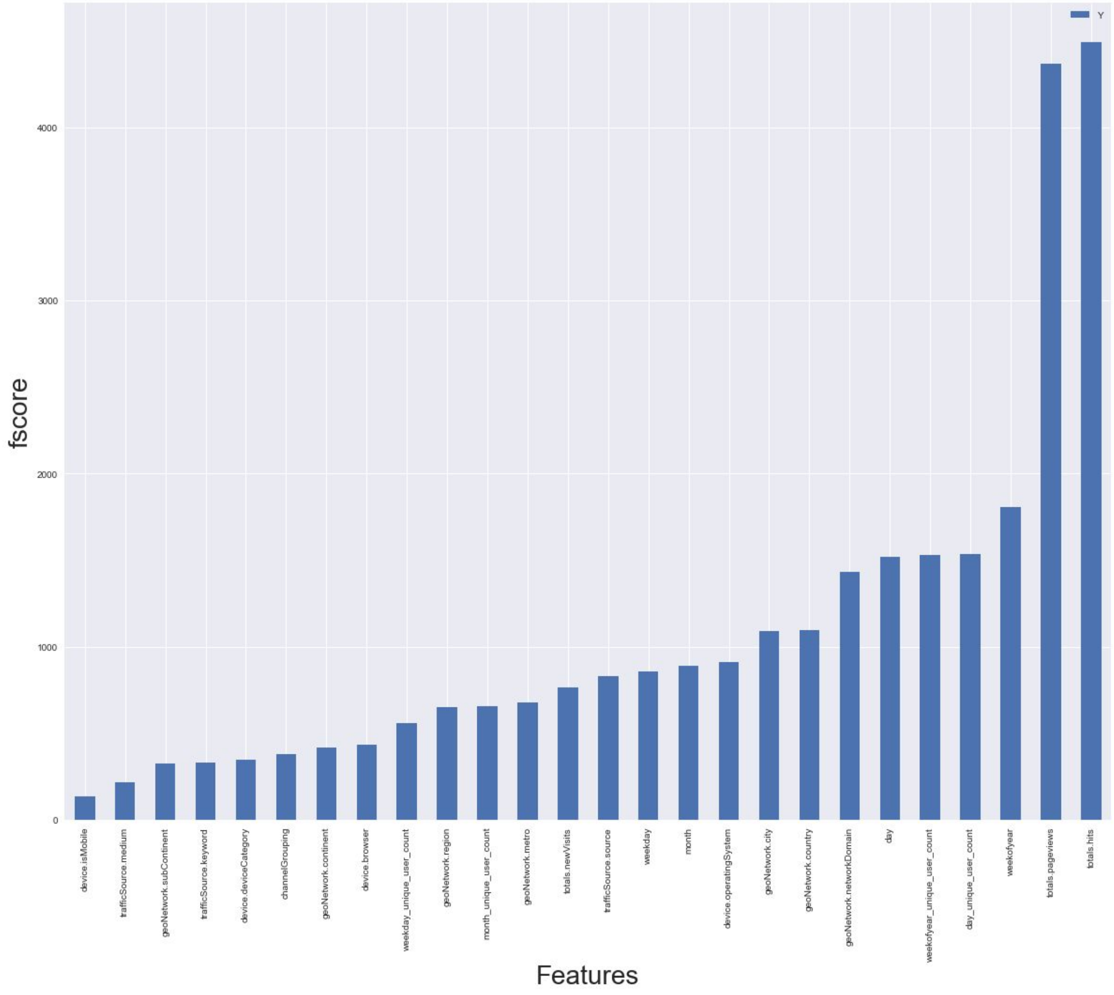
num_boost_round	train-rmse-mean	train-rmse-std	test-rmse-mean	test-rmse-std
982	1.490417	0.004174	1.626148	0.015321
983	1.490274	0.004129	1.626154	0.015336
984	1.490198	0.004124	1.626178	0.015348
985	1.490067	0.004142	1.626139	0.015353
986	1.489981	0.004145	1.626193	0.015340
987	1.489824	0.004160	1.626163	0.015365
988	1.489636	0.004189	1.626107	0.015327
989	1.489525	0.004214	1.626092	0.015329
990	1.489395	0.004193	1.626082	0.015356
991	1.489244	0.004183	1.626088	0.015371
992	1.489131	0.004193	1.626112	0.015378
993	1.488955	0.004132	1.626132	0.015402
994	1.488873	0.004144	1.626140	0.015424
995	1.488742	0.004144	1.626123	0.015419
996	1.488652	0.004168	1.626137	0.015413
997	1.488502	0.004227	1.626155	0.015436
998	1.488374	0.004221	1.626172	0.015444
999	1.488232	0.004247	1.626205	0.015445

This are the results of the cross validation score in 5 folds, we can see that the model keeps the good results across all the validation folds.

V. Conclusion

Free-Form Visualization

We did a feature imporcene graph at the end:



From this we can see the total page views and the total hits play a huge deal of importance, a lot of our engineered features are in the top 5 of relevance which made me very happy.

Reflection

I think the most interesting aspects of the project, were working with JSON fields, and the whole transformation from which I learned a lot, and thankfully the kagglers helped a lot with contributions, which was something that I was unaware as well, how easy is to learn by sharing and reading other people’s approaches, I for sure improve a lot on how Kaggle works and will keep on participating on competitions, more than the money is the great deal of knowledge you take from each.

Feature engineering and data preprocessing take a great deal of any ML problem and I did not expect that it came as a surprise, it makede learn a lot and really motivated me to learn much more about it, since the is not a lot of material on the nanodegree on the topic.

Learning about what the Bayesian Optimization worked was also a great deal for the project, my own limitations made me look for new techniques that were not included in this nanodegree. I am spoecually haooy and proud that the Bayesian optimization along with my feature engineering process worked greatly, I did not try anything cutting edge since I am not a pro yet, but to implement something totally new for me was really cool when it finally worked and it just made even more excited to try a bunch of new stuff.

Improvement

For sure a lot of improvements can be done in this project, one of the first that comes to my mind is to reduce even more the number of features in our dataset. The Bayesian optimizer works much better with less features and as we could see in the above graph a lot of them are quite useless.

Using heavier parameters in our bayesian optimization part would also improve the result, but as those go higher also does the training time and the computational cost on which I am short.

I am not really good a te feature negeree and a lot of competitors have used that to improve their models even further, if I had more knowledge in that regard I would mainly use it to improve our feature and reduce it, to less than 20 features, I also do not have a lot of experience using the lightGBM and tuning it with bayesian optimization seemed too complicated for me to doi, but a lot of the top kagglers are using irt for this competitions, so I would as well.

For sure there is a lot of improvement if we use our current guess as a new benchmark, as I said before, feature reduction, better feature engineering and using a different algorithm will sure improve the score.

REFERENCES.

[1] https://en.wikipedia.org/wiki/Pareto_principle , Pareto principle (20/80 rule) .

[2] https://www.kaggle.com/c/ga-customer-revenue-prediction/data , Google Analytics Customer Revenue Prediction dataset.

[3]https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/ , XGBoosting

[4] https://en.wikipedia.org/wiki/AdaBoost, AdaBoost

[5] https://es.wikipedia.org/wiki/Gradient_boosting, GradientBoosting